Scheduling Complex Computer Simulations on Heterogeneous Non-dedicated Machines: A Case Study in Structural Bioinformatics*

Marco A. S. Netto‡, Ardala Breda, Osmar Norberto de Souza
Programa de Pós-Graduação em Ciência da Computação - PPGCC
Faculdade de Informática - FACIN
Pontifícia Universidade Católica do Rio Grande do Sul - PUCRS
Porto Alegre, Brazil
{stelmar,abreda,osmarns}@inf.pucrs.br

Abstract

Complex computer simulations are a class of applications that demands high performance processing power in order to be realized in a feasible time. To achieve this processing power, networks composed of non-dedicated machines are increasingly being investigated. An efficient scheduling scheme is one of the most important issues to make a better use of these resources. In this paper we present an architecture for scheduling complex computer simulations aimed at heterogeneous non-dedicated machines which relies on information provided by the models that are being simulated. Furthermore, a case study demonstrates how the proposed architecture can assist in the execution of complex simulations applied to the protein structure prediction problem, which is one of the most important current challenges in structural bioinformatics.

1. Introduction

Computer clusters are a well-known alternative to supercomputers when high performance computing is needed. Although clusters provide a very good cost/benefit rate, they still are very expensive for some realities around the world. Another approach to achieve the processing power for executing complex applications is by exploiting idle CPU cycles in computer networks. This approach is becoming increasingly investigated due to the vast amount of physics, biological and chemical applications that can benefit from the massive computing power being wasted all over the world [18].

*This research was developed at PUCRS and supported by grants from CAPES, CNPq and FAPERGS to Osmar Norberto de Souza.

‡Marco is currently a PhD student, at Universidade de São Paulo, and member of InteGrade Project (http://gsd.ime.usp.br/integrade). InteGrade is supported by a grant from CNPq, process number 552028/02-9.

SETI@home [2], Folding@home [15], FightAIDS@-Home [7] and Screensaver Lifesaver [17] are examples of projects that use geographically distributed resources to solve complex problems. Although there are several applications that already benefit from these distributed resources, many fundamental challenges are still being investigated, such as scheduling and distribution of tasks, fault tolerance, heterogeneity, scalability and security [9]. One of the main issues to achieve an efficient use of distributed resources is the scheduling of the tasks that comprise the user applications.

In this paper we focus our work on the scheduling of independent simulations that are performed in heterogeneous non-dedicated machines. As computer simulations addressing the solution of complex problems are always in need of an ever increasing computing power, they are a very important class of applications that can benefit from the utilization of distributed environments composed of non-dedicated machines. The main goal of this work is to describe an architecture for scheduling computer simulation that relies on output files generated by the simulation tools and the ability of these tools to restart simulations from their interrupted states. In order to demonstrate the use of the proposed architecture, we provide some initial experimental results by using a case study applied to the protein structure prediction problem, which is one of the most important current challenges in structural bioinformatics. It is important to highlight that the proposed architecture is not to be compared to or substitute the well-known resource managers, such as Globus [8] or Condor [10]. Rather, it constitutes a scheduling module that can be incorporated in these software systems to be used in a grid computing environment.

The remainder of this paper is organized as follows: Section 2 presents some related work and the motivation for the proposed architecture; Section 3 describes the architecture, including the description of its modules and the information

flow among the architecture internals components; Section 4 describes a case study illustrating how the architecture can assist in the execution of computer simulations applied to the protein structure prediction problem, including preliminary results; and finally, Section 5 presents our conclusions and future work.

2. Related Work and Motivation

There are several ongoing projects related to the execution of tasks in distributed environments [1, 3, 6, 13, 16, 21]. Some of the available software systems support heterogeneous resources, specification of user requirements (e.g., an application must be executed on a Pentium IV with a RAM of 2GB and the GNU/Linux operating system) [6], execution of parameter sweep applications [1] and migration of tasks [13]. Moreover, there are statistical techniques available for estimating resources characteristics to improve the performance of the user applications [21], as well as software systems that perform scheduling by replicating tasks when there are more computer resources than tasks to be executed [16].

Although many software systems provide valuable functionalities, the current execution information of the user application are not usually exploited. Another possibility is to build the user applications on a framework that provides services for scheduling tasks and load balancing at the application level [3]. Although such an approach provides good performance results, the source code of most scientific applications is usually large and too complex to be easily modified, thus demanding time and human resources, which are often unavailable. Furthermore, depending on the case, users may not have access to the source code of the applications or they are unable to modify the source code due to license issues.

In the architecture proposed here the scheduling process is accomplished by heuristics based on monitoring information from the user simulations. The idea of this work is to use two important functionalities that are, in general, already available in the simulation tools: the ability of restarting a simulation from its interrupted state, through the use of checkpoint files, and the periodic generation of the output files. The checkpoint files can be used to restart a simulation if a computer becomes available, or to migrate a simulation to another computer. The output files provide relevant information on the current state of the simulating model, which can in turn be used to figure out whether a simulation is yielding or not the desired results. Using this information a decision is then taken on whether the simulation should continue in the same resource, migrated to another heterogeneous resource or be cancelled out. All these analyses and decisions are taken based on heuristics defined by the user.

The approach we are developing and presenting here is very important for an efficient investigation of complex scientific problems that demand the realization of multiple simulations of a single model in order to find the one that best describes its real system counterpart. As many of these simulations will produce undesirable results, they must be identified as early as possible along their execution in order to allow the modification of the simulation input parameters or the model itself, therefore minimizing the time spent in their search. For these reasons it is essential to secure that the best simulations are always being executed and, if possible, in the best available computer resources. Note that this architecture is not aimed at optimizing the execution time of all simulations provided by the users, but only those yielding the best results as specified by the heuristics.

3. Architecture Internals

The proposed architecture comprises two modules (Figure 1). The first one is the server module, which is responsible for scheduling and distributing the simulations. The server module is executed in the server machine, which constitutes a gateway between the users and the worker machines. The second one is the worker module, which is responsible for controlling the execution of the simulations. The worker module is executed in the machine that supplies the computing power. Communications occur only between the server and the workers, that is, the workers do not share information among themselves.

3.1 Worker module

The worker module is responsible for starting and stopping the simulations in the worker machines, as well as collecting information on the current state of the ongoing simulations. This module relies on two main components: the communication layer and the execution monitor (Figure 1).

The **worker communication layer** is the component that holds information about the worker, such as hostname, computer power and the identification of the simulation that it is performing. Its main goal is to provide a mechanism to exchange information between the worker and the server.

The **execution monitor** is a plugin defined by the user in order to filter the simulation output data written to disk so as to facilitate the scheduling procedure accomplished in the server. When the filtered data are available, the execution monitor notifies the communication layer to transfer the data to the server. In order to prevent unnecessary messages to be sent to the server, the worker communication layer verifies whether the data are different than the previous ones.

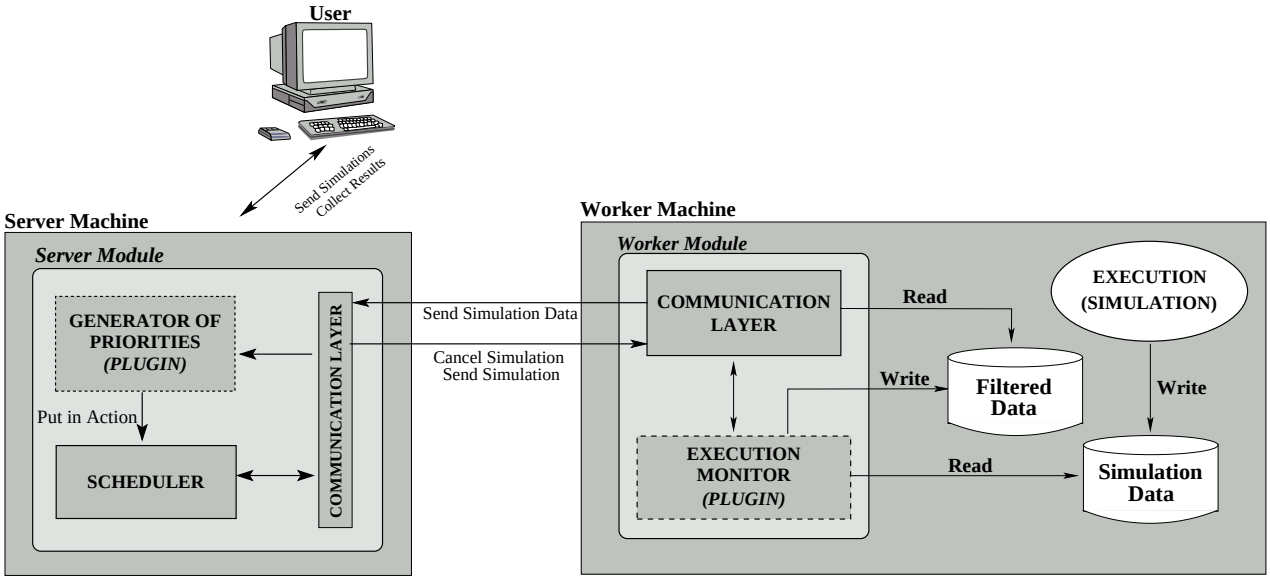

Figure 1. The architecture's components and their information flow.

3.2 Server module

The server module is responsible for scheduling and distributing the simulations among the workers. It consists mainly of three components, namely the scheduler, the generator of priorities and the communication layer (Figure 1).

The **server communication layer** is basically used to transfer information to and from the worker machines. Examples of information are: requests to cancel an ongoing simulation or start a new one, receive simulation data from the worker machines to perform the scheduling procedure, as well as keep-alive messages to verify the workers' availability.

The **generator of priorities (GP)** is a plugin built by the user, that relies on an Application Programming Interface (API) which provides two functions: update the data from a simulation and, based on these data, generate the priorities of the simulations. Applying the heuristics, defined in this component, the filtered data received from the workers are converted to priority values. When a priority of a simulation is modified, the scheduler is put in action.

With the priorities of the simulations available, the **scheduler** decides where each simulation must be executed. Note that the assignment of the simulations to the workers may involve the migration of a simulation from a resource to another, as well as the temporary interruption of an ongoing simulation.

In order to perform the correct scheduling, it is necessary to verify the workers' availability that are supposedly executing simulations. This verification becomes necessary mainly because the architecture was designed to support non-dedicated machines, therefore faults are a rule and not an exception. The goal of the algorithm used in the scheduler is to assign the best workers to the simulations that are yielding the best results, that is, those that hold the higher priorities. Considering that the workers join and leave the network and the priorities of the simulations are modified during their execution, the scheduling is dynamically performed. The verification of the workers' availability must be effected before the scheduling of the simulations, as well as at time intervals in order to keep all the simulations executing. Thus, if the server detects that a simulation was interrupted due to its worker is not more available, the server tries to assign this simulation to another worker, whenever possible.

3.3 Implementation of the architecture

The architecture described here is already implemented using the C language. We have also implemented a simulator of the architecture where the main difference to the software of the architecture is that the simulator distributes the computer simulations to a local machine. In addition to the functionalities described in this paper, others were included in the simulator, such as the simulation of the workers' availability. Instead of users having to include or remove the workers randomly and at any time, an option was implemented to make possible to define workers and their availability according to periods of day.

In the current implementation of the architecture we assume the server and worker machines as having the same view of the file system, hence turning the distribution of the simulations considerably simple. The distribution basically involves the transference of a file which has the paths to the

following inputs: the simulation input files, the simulation tool and the script or program that implements the execution monitor.

4 Case Study: Molecular Dynamics Simulations of Proteins

One of today's challenges following the post-genomic era, that is, the high-throughput sequencing of genomes, is the conversion of these data into useful information such as translation of nucleotide sequences into known genes and, ultimately, to proteins with a characterized structure and function [14].

Not all proteins can have their three-dimensional (3D) structure known by experimental methods. Thus, new methods for protein structure determination become necessary, and that is one of the major goals of structural bioinformatics, the development and utilization of computational tools to characterize these proteins, including the prediction of their 3D structures, in order to infer hypotheses about their functions [14, 19].

There are different computational approaches to predict protein structure [20]; herein we focus on the molecular dynamics (MD) method. MD is based on the principles of classical mechanics and can give a microscopic view of the behavior of each atom that composes a system like a protein [12]. In a MD simulation, the dynamic behavior of a protein is provided, and once the given ensemble (the group of structures generated along the simulation) reaches the equilibrium, parameters such as the average structure can be calculated [5].

We are currently developing MD protocols to enable the correct prediction of any protein 3D structure, using only the linear sequence of amino acids of a protein as the initial information. Figure 2 illustrates a set of snapshots generated by one of the simulations of the polyalanine 12 (PA12), the model protein[1] used in our experiments, as well as the reference structure, which is the one we desire to achieve. As a means to standardize our MD protocols we experiment with different parameters available from AMBER [4], a well-known set of computer programs aimed at performing MD simulations of biomolecules.

One approach to simulate our models is by executing the simulations one by one on a cluster, since AMBER supports Message Passing Interface (MPI) [11]. Another approach is to execute the simulations on networks composed of non-dedicated machines. Due to the amount of exchange information necessary among the processes to execute the simulations using the MPI version of AMBER, the easiest way to perform the simulations is by executing each different simulation on a single computer.

[1] Our model protein is a polypeptide of 12 alanine amino acids.

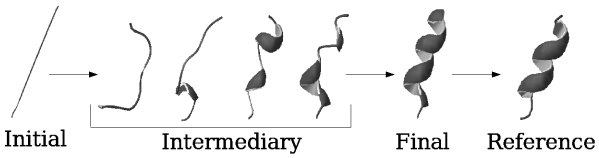

Figure 2. Examples of snapshots generated by a MD simulation of PA12.

In order to improve the utilization of the computing power available, we developed an architecture aimed at prioritizing the simulations that provide the best results. The challenge in using this architecture now lies in choosing the best simulations on-the-fly. For this reason, the model to be simulated and the output files provided by the simulation tool must be analyzed in details to build efficient plugins, that is, the execution monitor and the generator of priorities.

To evaluate the architecture, we have simulated a small model protein for 2 nanoseconds (biological process time). Different results were generated by performing 15 simulations of the same protein, varying two input parameters. The first parameter is the final temperature of the protein, and the second one is the seed number to generate the initial velocities of the atoms. In AMBER, a simulation can be executed in packages so as to allow it to be restarted from the last simulated package if it is interrupted. Thus, the migration process is performed by canceling the current execution of a simulation and restarting it from the last completed package. In our experiments, we defined 20 packages, each one corresponding to 100 picoseconds.

4.1 Choosing the best simulations

In order to schedule the simulations, two procedures must be performed: (i) determine parameters that define if a simulation is yielding satisfactory results and (ii) specify how to use these parameters. The first task is performed by the execution monitor and the second one by the generator of priorities.

To choose the best of a set of protein simulations we have used the root mean square deviation (RMSD), a parameter that shows how similar, or even how different, two structures are; herein the simulated trajectory is compared to the experimentally refined structure. Plotting the RMSD versus time, the evolution of the protein can be seen, from the extended/non-native state (higher RMSD values) to its native, biologically functional state (lower RMSD values). With these values two different structures, in this case the simulated and the reference one, can be compared. The lower the RMSD, the better is the simulation, being therefore favored in the scheduling process. The RMSD can be

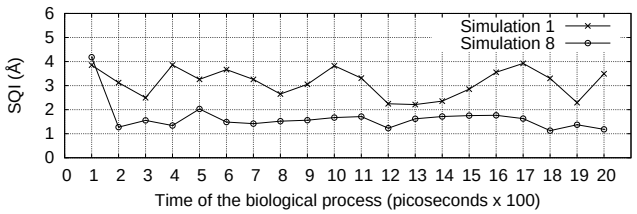

Figure 3. Behavior of the SQI in relation to the time of two simulations.

obtained by filtering the output files generated by AMBER while the simulations are being executed. In AMBER, for each picosecond simulated (biological process time), a new RMSD becomes available. However we decided to create a new parameter, called **Simulation Quality Index (SQI)**, which represents the average of the 100 RMSDs of a package in addition to its standard deviation. Thus, when a package becomes completed, the worker sends to the server the SQI of its simulation.

The next step is the definition of how to use the SQI, which is done in the generator of priorities. In our experiments we have developed three GPs, GP1, GP2 and GP3, that were built to work with three, four and eight priorities respectively:

1. GP1:

$$Priority = \begin{cases} 3 & : \quad 0 < SQI \leq 2 \\ 2 & : \quad 2 < SQI \leq 3 \\ 1 & : \quad SQI > 3 \end{cases}$$

2. GP2:

$$Priority = \begin{cases} 4 & : \quad 0 < SQI \leq 1.5 \\ 3 & : \quad 1.5 < SQI \leq 2.5 \\ 2 & : \quad 2.5 < SQI \leq 3.5 \\ 1 & : \quad SQI > 3.5 \end{cases}$$

3. GP3:

$$Priority = \begin{cases} 8 & : \quad 0 < SQI \leq 1.4 \\ 7 & : \quad 1.4 < SQI \leq 1.6 \\ 6 & : \quad 1.6 < SQI \leq 1.8 \\ 5 & : \quad 1.8 < SQI \leq 2.0 \\ 4 & : \quad 2.0 < SQI \leq 2.25 \\ 3 & : \quad 2.25 < SQI \leq 2.5 \\ 2 & : \quad 2.5 < SQI \leq 3.0 \\ 1 & : \quad SQI > 3 \end{cases}$$

It is expected that some simulations will have their priorities more varied than others, thus generating excess migrations. Figure 3 illustrates the behavior of SQI of each simulated package in relation to the biological time of two simulations: Simulation 8, which yields desirable structures since its SQI seldom oscillates about 1.5 Å, and Simulation 1, which yields undesirable results because its SQI oscillates frequently.

In order to implement the GPs, the two API functions available in the architecture were used (Section 3.1). The first one basically involves the update of the SQI values of a simulation when a package becomes completed. In this function the user only needs to declare the variable SQI to be updated. The second one is the function responsible for generating the priority of a simulation, which consists of a set of conditional statements to compare the SQI value of a simulation with the user defined ranges.

4.2 Description of the experiments: simulating a protein

The experiments were executed using the simulator, as well as the architecture in a real network. For both environments, the network used to perform the simulations was composed of 15 workers with three different computing power. In relation to the workers' availability, it was assumed that, in a period of four hours, during three hours all workers were available and during one hour only 50% of them were available.

To perform the experiments in the simulator, the output files of the protein simulations previously realized were used. This method was adopted since the execution of all simulations in a single machine at the same time would not be possible due to the computing power limitation. Therefore, the protein simulations were performed one by one in a single machine and all the output files were saved. Using these files, the SQI values were calculated and filtered, at 100 picoseconds time intervals, for each simulation. To simulate the behavior of a simulation in relation to the generation of the output files, it was implemented a simple program to read the file that has the list of all SQIs and store the last SQI read in an output file. This program can also be interrupted and restarted from the last completed step (or package in a real simulation), as well as to generate the output files using a frequency according to the worker computing power. In order to be able to perform several experiments in a short period of time, the time scale in the simulator was reduced 50 times. That is, each hour in the simulations represents 72 real seconds.

The real environment where the experiments were performed is composed of 15 heterogeneous machines interconnected by a Fast-Ethernet network[2]. The only difference among the machines are their processors, which are Pentium 32 bits with three distinct processor frequencies.

[2]Machines managed by Research Center in High Performance Computing - CPAD - http://www.cpad.pucrs.br.

Table 1. Completion order of the simulations.

Simulation	$\overline{SQI}$ (Å)	Simulator				Real Environment			
		without GP	GP1	GP2	GP3	without GP	GP1	GP2	GP3
8	1.7 ± 0.6	8	3	2	3	10	7	3	1
7	1.7 ± 0.8	7	2	1	1	6	2	2	2
6	1.8 ± 0.7	6	1	5	5	13	1	4	5
13	1.9 ± 0.4	10	6	6	6	11	3	11	4
5	1.9 ± 0.6	5	4	3	4	5	4	1	6
14	1.9 ± 0.8	11	8	9	7	9	6	8	7
15	2.1 ± 1.0	12	5	4	2	12	5	5	3
12	2.5 ± 0.7	9	9	7	8	8	15	7	8
4	2.6 ± 0.9	4	7	8	10	4	8	6	12
10	2.9 ± 0.6	13	10	13	14	14	9	13	13
11	3.0 ± 0.7	14	12	12	12	15	12	12	14
3	3.0 ± 0.8	2	11	10	9	1	10	9	9
1	3.1 ± 0.6	3	14	11	15	2	13	10	10
9	3.2 ± 0.7	15	15	15	13	7	14	15	15
2	3.3 ± 0.6	1	13	14	11	3	11	14	11

4.3 Experimental results

Table 1 illustrates the order in which the simulations were completed in the experiments performed both in the simulator and in the real environment. The data are sorted by the average of the Simulation Quality Index ($\overline{SQI}$) of all 20 simulated packages. When no priorities were used, the simulations were completed without an order. Therefore, in order to discover the best simulations, that is, those with lower $\overline{SQI}$ values, all simulations must be completed. Using GPs, the best simulations were completed before the worst ones, hence achieving the expected goal. However, using them means to incorporate a considerable cost due to the execution wasted by migrations.

Table 2 summarizes the costs generated by using GPs in relation to the execution of the simulations without GPs. The costs were analyzed using three parameters. The first one is the amount of **Execution Time (ET)** for all simulations. The ET of a simulation begins when a simulation is submitted to the server and finishes when the simulation is completed. The second parameter is the **Number of Migrations (NM)** occurred during the execution of all simulations. The NM represents the number of times a simulation was migrated from a worker to another, as well as when a simulation is interrupted due to its worker becomes unavailable, and after that, it is assigned to another worker or to the same one. The last parameter is the **Wasted Time (WT)** of all simulations. WT measures the wasted time of an execution when a simulation is interrupted during the execution of a package. This interruption may occur in two cases: (i) the simulation worker becomes unavailable; or (ii) a simulation with a higher priority needs its worker. The data show that the cost of using GPs increases with the number of priorities and the size of the SQI ranges specified in each GP.

Although there exists a considerable cost when GPs are used, the data presented in Table 2 take into account the execution of all simulations of PA12. It is possible to create another scenario where it is not necessary to execute all simulations. In case a user, for example, needs only some simulations that satisfy a determined criterion, the use of GPs becomes very important. The following hypothesis illustrates such a scenario.

Table 2. Costs generated by using the GPs.

Environment_GP	ET (%)	NM (%)	WT (%)
simulator_GP1	12.96	440.00	222.46
simulator_GP2	13.72	440.00	215.08
simulator_GP3	18.70	696.00	318.77
real_GP1	9.55	372.00	226.34
real_GP2	11.99	420.00	285.60
real_GP3	17.91	724.00	399.18

Let us consider a user that needs only N simulations that generate 3D structures with $\overline{SQI}$ lower than or equal to 2.0 Å. Figures 4a and 4b illustrate the total execution time of the N simulations that satisfy the criterion established by this user. In the experiments realized in the simulator, GP1 is the best in all cases. Regarding the experiments realized in the real environment, GP1 and GP2 are the most efficient, with the exception of GP3 when N is 5. Since GP1 is the best GP in three cases (N = 1, 2 e 6), and GP2 in two cases (N = 3 e 4), then GP1 is the most indicated to perform the

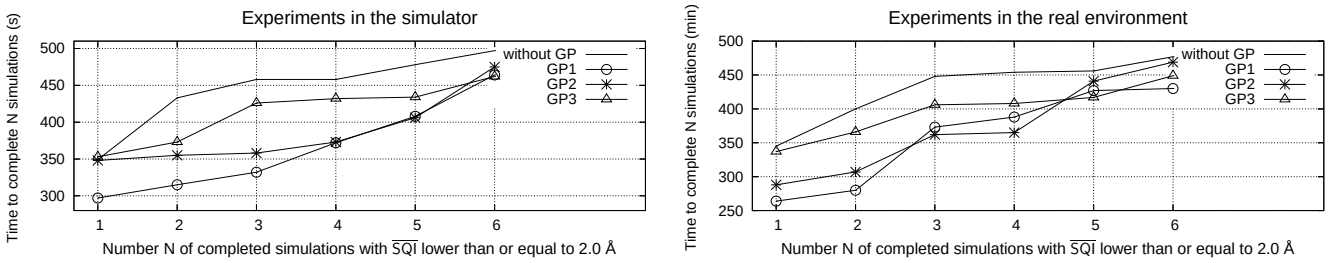

Figure 4. Efficacy of the scheduling when the user needs only the best simulations.

simulations in the real environment. In this example, we can conclude that the utilization of GPs is important when there is no need to complete all simulations. The advantage of this approach is that the users can have their results earlier and, consequently, are able to execute new simulations of other models, or even to execute simulations of the same model but with new input parameters.

It is important to highlight that the GPs were built knowing that PA12 holds a structure of easy prediction. If we did not have a good knowledge about PA12, we could create GPs with SQI ranges higher or lower than those defined in the experiments, which would not effect the use of priorities. Therefore, the challenge of using the proposed architecture is in the building of good generators of priorities through the knowledge of the simulated models.

5 Concluding Remarks and Further Work

In this paper we have presented an architecture aimed at scheduling complex computer simulations on environments composed of heterogeneous non-dedicated machines. The proposed architecture has been designed to allow the best simulations to be executed first and in the best machines. In order to achieve this goal, the architecture utilizes the files generated by these simulations, allowing the users to automatically prioritize the simulations without modifying the simulation tools or linking them to any library. The user only needs to build two simple plugins: one to filter the simulation data and the other to use the filtered information to determine the simulation priorities. Furthermore, the architecture has been implemented in order to be the most generic as possible in the sense that several simulation tools could be utilized, and, hence, many researchers could perform their complex simulations in environments composed of non-dedicated machines.

From the initial experiments based on a well-known structural bioinformatics problem, it could be observed that the architecture has achieved its expected purpose. However, several improvements are in course. Some of them are:

- **Partial execution of the simulations**: in our experiments we have observed that some simulations become stable after a short period of time. Thus, it is interesting to incorporate into the GPs stop criterion for the simulations in order to allow new ones start executing as soon as possible;
- **Dynamic generator of priorities**: the construction of good GPs is fundamental to the efficient use of the resources. However, the users, when simulating their models, may not have any idea about the behavior of their simulations. Therefore, a very useful approach to solve this problem is to build GPs dynamically and automatically using the information about the ongoing simulations. When this component becomes available, the only task the user will need to take care of is the definition of parameters to be used in GPs;
- **Implementation of the plugins**: currently the GP must be implemented in C language, and the execution monitor also in C or in a script language. We are studying alternatives to make easier the development of the architecture plugins. This is particularly important because we desire that researchers with no expertise in computer science could benefit from this architecture to perform their simulations;
- **Minimizing migrations**: we are studying some approaches to minimize the amount of work lost due to migrations. For example, one approach is to define that a simulation closed to generate checkpointing files could not be migrated even if a simulation with a higher priority needs its worker. Another approach is to automatically define the frequency in which checkpointing files are generated in the simulation tool. Higher frequencies may minimize the amount of wasted work, however, the amount of bytes to be stored and transfered may increase;
- **Computational Grid infra-structure**: the proposed architecture was not designed to be a complete computational grid infra-structure to provide functionalities of security, transference of files and user graphical

interface. For this reason, the architecture developed here must be adapted in a robust grid infra-structure, such as Globus [8] or Condor [10], in order to incorporate more functionalities. In this context, one important issue is the scalability, since the migrations of simulations may increase the execution time. One possible approach is to execute the simulations in groups of machines with a good connectivity, and thus allowing the migrations only among these machines.

The architecture presented in this work is related to the scheduling of applications in distributed systems. In computational grids, the scheduling of applications is a complex problem due to the heterogeneity and the dynamicity of the resources. Therefore, this work has illustrated an alternative in which it is possible to use the applications information, in this case computer simulations, to scheduling them without modifying the simulation tools.

Structural bioinformatics is one of the promising areas in need of utilizing the available resources in computational grids. The prediction of protein 3D structure from sequence alone is particularly important since the knowledge of these structures can be used to design new or alternatives drugs for the treatment of diseases. Therefore, in this context, the work presented here may be used to assist scientists to make a better use of computing resources through an efficient schedule schema of their simulations.

References

[1] D. Abramson, R. Sosic, J. Giddy, and B. Hall. Nimrod: A Tool for Performing Parameterised Simulations using Distributed Workstations. *Proceedings of Fourth IEEE International Symposium on High Performance Distributed Computing*, pages 112–121, 1995.

[2] D. P. Anderson, J. Cobb, E. Korpela, M. Lebofsky, and D. Werthimer. SETI@home: an experiment in public-resource computing. *Communications of the ACM*, 45(11):56–61, 2002.

[3] F. Berman, R. Wolski, H. Casanova, W. Cirne, H. Dail, M. Faerman, S. Figueira, J. Hayes, G. Obertelli, J. Schopf, G. Shao, S. Smallen, N. Spring, A. Su, and D. Zagorodnov. Adaptive Computing on the Grid Using AppLeS. *IEEE Transactions on Parallel and Distributed Systems*, 14(4):369–382, 2003.

[4] D. A. Case, D. A. Pearlman, J. W. Caldwell, T. E. C. III, J. Wang, W. S. Ross, C. L. Simmerling, T. A. Darden, K. M. Merz, R. V. Stanton, A. L. Cheng, J. J. Vincent, M. Crowley, V. Tsui, H. Gohlke, R. J. Radmer, Y. Duan, J. Pitera, I. Massova, G. L. Seibel, U. C. Singh, P. K. Weiner, and P. A. Kollman. Amber 7. *University of California, San Francisco*, 2002.

[5] O. N. de Souza and R. L. Ornstein. Molecular dynamics simulations of a protein-protein dimer: particle-mesh ewald eletrostatic model yields far superior results to standard cutoff model. *Journal of Biomolecular Structure and Dynamics*, 16:1205–1217, 1999.

[6] D. W. Erwin. UNICORE - A Grid Computing Environment. *Concurrency and Computation: Practice and Experience*, 14(13-15):1395–1410, 2002.

[7] FightAIDS@home. http://www.fightaidsathome.org. Accessed on February 2005.

[8] I. Foster and C. Kesselman. Globus: A Metacomputing Infrastructure Toolkit. *The International Journal of Supercomputer Applications and High Performance Computing*, 11(2):115–128, 1997.

[9] I. Foster and C. Kesselman. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan-Kaufman, San Francisco, CA, 1999.

[10] J. Frey, T. Tannenbaum, I. Foster, M. Livny, and S. Tuecke. Condor-G: A computation management agent for multi-institutional grids. *Cluster Computing*, 5:237–246, 2002.

[11] W. Gropp, E. Lusk, N. Doss, and A. Skjellum. A high-performance, portable implementation of the MPI message passing interface standard. *Parallel Computing*, 22(6):789–828, 1996.

[12] W. F. V. Gunsteren and H. J. C. Berendsen. Computer simulation of molecular dynamics: methodology, applications, and perspectives in chemistry. *Angew. Chem. Int. Ed. Engl*, 29:992–1023, 1990.

[13] M. Litzkow, M. Livny, and M. Mutka. Condor - a hunter of idle workstations. *Proceedings of the 8th International Conference of Distributed Computing Systems*, pages 104–111, 1988.

[14] N. M. Luscombe, D. Greenbaum, and M. Gerstein. What is Bioinformatics? A Proposed Definition and Overview of the Field. *Methods in Medical Informatics*, 40:346–258, 2001.

[15] V. S. Pande, I. Baker, J. Chapman, S. Elmer, S. M. Larson, Y. M. Rhee, M. R. Shirts, C. D. Snow, E. J. Sorin, and B. Zagrovic. Atomistic protein folding simulations on the submillisecond time scale using worldwide distributed computing. *Peter Kollman Memorial Issue, Biopolymers*, 68(1):91–109, 2003.

[16] D. Paranhos, W. Cirne, and F. Brasileiro. Trading Cycles for Information: Using Replication to Schedule Bag-of-Tasks Applications on Computational Grids. *Proceedings of the Euro-Par 2003: International Conference on Parallel and Distributed Computing*, pages 169–180, 2003.

[17] W. G. Richards. Virtual screening using grid computing: the screensaver project. *Nature Reviews Drug Discovery*, 1:551–555, 2002.

[18] M. Shirts and V. Pande. Screen savers of the world, unite! *Science*, 290(8):1903–1904, 2000.

[19] D. Shortle. Prediction of Protein Structure. *Current Biology*, 10:R49–R51, 2000.

[20] C. M. Smith. Molecular modeling in the genomics era. *The Scientist*, 15(5), 2001.

[21] R. Wolski. Experiences with predicting resource performance on-line in computational grid settings. *ACM SIGMETRICS Performance Evaluation Review*, 30(4):41–49, 2003.